METHOD AND SYSTEM FOR RE-TARGETING INTEGRATED CIRCUITS

FIELD OF THE INVENTION

[0001] The invention relates generally to the field of electronic design automation systems for digital logic, and more particularly to a method and system for allowing a designer to re-target integrated circuit of different vendors from a set of post routed boolean equations targeted to an integrated circuit of one vendor.

BACKGROUND OF THE INVENTION

[0002] Due to advancements in processing technology, it is now possible to pack millions of transistors in an integrated circuit (IC). As a result, very large and complex circuits can be implemented into an IC. This means that it is extremely difficult for a design engineer to design ICs manually. Currently, almost all design engineers use a broad range of software tools to generate and analyze models of an intended system prior to its fabrication. These tools support the efficient generation of circuit implementation details from abstract specification models. They are also able to detect design mistakes early in the design process, thereby reducing unnecessary efforts spent on erroneous designs. These tools contribute to great savings in time and money needed to develop ICs.

[0003] During the early development of electronic design automation, designers define the various modules of an IC at the gate level using a schematic capture and/or simple Hardware Description Language (HDL) technique. Evolution of HDLs and advances in simulation and synthesis technologies have enabled IC designers to design at a higher level (register transfer level, or RTL) than at the gate level.

[0004] When designing using a HDL, the designer describes a module in terms of signals that are generated and propagated through combinatorial modules from one set of

EL620970425US

registers to another set of registers. HDLs provide a rich set of constructs to describe the functionality of a module. HDLs allows designs to be described at different levels. At a high level, the functionality is described using high level constructs such as ALWAYS blocks with sensitivity list, IF statements, CASE statements, and procedural continuous assignment statements that use logic operators, arithmetic operators, and relational operators. For example, Verilog and VHDL allow designs to be described at the behavioral, register transfer, and at the gate level. Similarly, ABEL allows designs to be described at the register transfer level (RTL) and at the structural level. There are a number of HDLs in addition to Verilog, VHDL, and ABEL. Some are proprietary to a single commercial vendor. A more detailed description of Verilog is set forth in Thomas and Moorby, "The Verilog Hardware Description Language," Kluwer Academic Publishers (1990). A more detailed description of VHDL is set forth in K.C. Chang, "Digital Design and Modeling with VHDL and Synthesis," IEEE Computer Society Press (1997). A more detailed description of ABEL is set forth in "ABEL-HDL Reference," Data I/O Corporation (1994).

[0005] After a HDL design is defined, the designer uses design software supplied by an IC vendor to optimize the design entry source and then place and route the design into that vendor's target IC. The design software also generates a set of optimized and post-routed low level boolean equations which represent the design functionality. These equations may be hardware dependent or independent. The equations are not written in a standard HDL, such as Verilog or VHDL. Instead, they are generally written using a proprietary language format of the vendor.

[0006] Fig. 1 is a diagram that summarizes the above-described prior art electronic design system. A chip designer generally defines a design using VHDL 32, Verilog 34, schematic 36 or mixed schematic/HDL 38. The design is then run through a design software 40 to target a specific device, generally a CPLD (complex programmable logic device), FPGA (field programmable gate array), gate array, standard cell based ASIC, or a standard custom designed integrate circuit product. Conventionally, this software reads the design entry source code (e.g., VHDL, Verilog, or schematics), optimizes the logic, and does place and route on the design. This design software can generate either vendor dependent or vendor independent set of boolean equations 42 into a file.

[0007] In the programmable logic device (PLD) industry, some design software supplied by a first PLD vendor can read low-level boolean equations written by the software of a second vendor, and then compile the equations to target devices of the first vendor. This is generally done to allow a designer who has originally targeted his/her design to the technology of the second vendor the ability to re-target the same design into a technology supported by the first vendor. One of the problems is that when designs are re-targeted from one vendor to another using this method, the designer cannot read the intermediate files generated by the design software, edit the design, or simulate the design.

[0008] Mixed schematic/HDL designs present additional difficulties. They contain top level schematics having user-defined block symbols for the HDL modules only. Typically, the schematics are vendor dependent, and are written using a proprietary schematic editor and proprietary symbol library of a vendor. As a result, they cannot be re-targeted to another family of ICs marketed by another vendor.

SUMMARY OF THE INVENTION

[0009] The present invention provides a novel method and system of electronic design. It allows a designer to easily re-target a design optimized for the device of one IC vendor to the device of another vendor. The designer can start with a set of post-routed boolean equations optimized for a certain target. A corresponding synthesizable, editable, and simulatable HDL description is generated. The designer may edit the HDL code. Another target may be selected. Design optimization and placement and routing can be performed for the new target.

[0010] To implement the present invention, a system for automatically generating a HDL description of a design from low-level boolean equations is provided. This HDL description is synthesizable, editable, and simulatable. The designer is no longer required to manually re-write the design into HDL code.

[0011] The above summary of the present invention is not intended to describe each disclosed embodiment of the present invention. The figures and detailed description that follow provide additional example embodiments and aspects of the present invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] Fig. 1 is a diagram showing a prior art electronic design system.

[0013] Fig. 2 is a block diagram showing a development environment of the present invention.

[0014] Fig. 3 is a flow chart showing a procedure of the present invention.

[0015] Fig. 4 is a flow chart showing the steps of generating HDL code of the present invention.

[0016] Fig. 5A shows various entities of low level boolean equations that can be used in the present invention.

[0017] Fig. 5B is a flow chart showing the parsing of logic equations in accordance of the present invention.

[0018] Fig. 6A shows various synthesizable VHDL objects in accordance with the present invention.

[0019] Fig. 6B shows various synthesizable Verilog objects in accordance with the present invention.

[0020] Fig. 6C shows various synthesizable ABEL objects in accordance with the present invention.

[0021] Fig. 7 is flow chart showing the steps for generating a VHDL code of the present invention

[0022] Fig. 8A is an example of a 4-bit counter described using logic equations in according with the present invention.

[0023] Figs. 8B-1 and 8B-2, in combination, show VHDL code corresponding to the 4-bit counter of Fig. 8A.

[0024] Figs. 8C-1 and 8C-2, in combination, show Verilog code corresponding to the 4-bit counter of Fig. 8A.

[0025] Fig. 8D shows ABEL code corresponding to the 4-bit counter of Fig. 8A.

[0026] Figs. 9A-9E, combined, show the steps for generating a Verilog code of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0027] The present invention relates to an innovative method to generate high-level hardware description language code. In the following description, numerous specific details are set forth in order to provide a more thorough understanding of the present invention. However, it will be apparent to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known features have not been described in detail in order to avoid obscuring the present invention.

[0028] As explained above, many designers prefer to work with high level techniques (such as HDL) in designing ICs. The scope of describing designs in HDL is quite large, and a rich set of constructs and functionalties are available to describe any design. However, only a subset of these constructs are synthesizable. That is, only a subset of all of the available constructs can be taken through automated synthesis tools (such as the Design Compiler that is commercially available from Synopsys, Mountain View, CA) to generate gate level netlist in terms of the cells from a target technology library provided by the semiconductor vendor. This limited set of synthesizable constructs is referred to herein as synthesizable set, and members of this set is called synthesizable objects.

[0029] Many designers like to work with synthesizable HDL constructs that can be easily targeted to ICs of different vendors. One aspect of the present invention is that it is able to identify the synthesizable objects in a set of boolean equations and translates them into synthesizable, editable, and simulatable HDL constructs. With the present invention, the designer does not need to know the syntax of the boolean equations to understand the functionality of the design.

[0030] The present invention operates in a development environment 100 as shown in Fig. 2. Environment 100 contains a computer system that includes a conventional processor 102, a user interface 104, and memory 110. In the preferred embodiment, a combination of non-volatile memory (such as hard disk) and volatile memory (such as semiconductor random access memory) are used. Memory 110 is used to store a plurality of software modules, such as a language compiler/linker 112, source code (generated by the user) 114, library files 116, object code 118, boolean equations 120 and a synthesizable and simulatable HDL code generator 122. Note that not all of these modules need to be present simultaneously. Language compiler/linker 112 and library files 116 may be conventional software packages that are readily available from various vendors, such as Sun Microsystems, Mountain View, CA, and Free Software Foundation, Boston, MA (which distributes the GNU software).

[0031] In one embodiment, the processor of Fig. 2 is a microprocessor of a UNIX workstation. A graphical user interface, such as X-windows, from the Massachusetts Institute of Technology, is used as the user interface. Alternatively, a system based on the so-called "X-86" architecture and the Microsoft Windows operating system (including Windows 95, Windows 98, Windows 2000, and Windows

NT) may be used. Software in memory 120 may be written in standard programming language, such as C++ and/or PERL.

[0032] The present invention provides a procedure of electronic design that is different from the prior art. Fig. 3 shows a procedure 140 of the present invention. It starts with a set of low level boolean equations (step 142). The boolean equations typically represent inputs, outputs, inouts, logic expressions, relational expressions, arithmetic expression, tristates, and registers. Based on this set of equations, a synthesizable, editable, and simulatable HDL source code is generated (step 144). The user may edit the code (step 146). Design optimization and placement and routing can be performed for a target IC selected by the user(step 148). A different set of low level boolean equations may be generated as a result of step 148.

[0033] The details of step 144 are shown in Fig. 4. In step 162, the low-level boolean equations are read into temporary storage (e.g., in volatile memory). In step 164, the equations are carefully parsed. In step 166, equations that give rise to synthesizable objects are identified. In step 168, information that is relevant to the synthesizable objects is obtained from the boolean equations. In step 170, the corresponding HDL code is generated.

[0034] Fig. 5A is a diagram showing various entities that may be present in boolean equations. The objects are a set of boolean functions commonly used in digital design. This set of boolean functions include signals 188, inputs, outputs, inouts (shown as 187), logic expressions 189, arithmetic equations 190, relational equations 191, registers, tristate buffers 194, and open drain outputs 195. The register elements include various forms of D flip flops 192 and T flip flops 193.

[0035] Fig. 5B is a flow chart 300 showing the parsing of low level boolean equations in accordance with the present invention. In step 302, the filename of the boolean equations is read. In step 304, the inputs, outputs and inouts are read. In step 306, signal declarations are read. In step 308, logic expressions are read. In step 310, other expressions (arithmetic and relational) are read. In step 312, D-type flip flops are read. In step 314, T type flip flops are read. In step 316, tristate buffers are read. In step 318, open drain outputs are read. The parsing of the boolean equations is completed.

[0036] Fig. 6A is a diagram showing various synthesizable objects in the synthesizable, editable, and simulatable VHDL language of the present invention. They include entity/architecture ports 205, input/output inout declarations 206, signal declarations 207, conditional D/T flip flop components 208, signal assignments 209, conditional D/T flip flop instantiation 210, design statistics 211, comments 212 and tristate/open drain outputs 213.

[0037] Fig. 7 is flow chart 330 showing the steps for generating VHDL code of the present invention. In step 332, information about design and/or statistics is written. In step 334, the inputs, outputs and inouts are written. In step 336, signal declarations are written. In step 338, combinatorial expressions are written. In step 340, D-type flip flops are written. In step 342, T type flip flops are written. In step 334, other combinatorial expressions are written. In this flow chart, other expressions include expander equations (an architecture specific resource on CPLDs) and equations which contain active low left hand side (LHS) arguments. In step 346, tristate buffers are read. In step 348, open drain outputs are written. The VHDL code is generated.

[0038] As an example, a 4-bit counter (described using boolean equations in according with the present invention) is shown in Fig. 8A. Figs. 8B-1 and 8B-2, in combination, provide the corresponding synthesizable and simulatable VHDL code.

[0039] The above principle can be applied to Verilog. Fig. 6B is a diagram showing various objects in the synthesizable, editable, and simulatable Verilog language of the present invention. They include module ports 235, input/output inout declarations 236, wire/reg declarations 237, conditional D/T flip flop module 238, procedural continuous assignments 239, conditional D/T flip flop instantiation 240, design statistics 241, comments 242 and tristate/open drain outputs 243.

[0040] Steps similar to that illustrated in Fig. 7 may be used to generate Verilog code for a corresponding set of boolean equations. Figs. 9A-9E, combined, show a flow chart illustrating detailed steps. Figs. 8C-1 and 8C-2, in combination, show the synthesizable and simulatable Verilog code corresponding to the 4-bit counter of Fig. 8A.

[0041] The above principle can be applied to ABEL. Fig. 6C is a diagram showing various objects in the synthesizable, editable, and simulatable ABEL language of the present invention. They include module ports 265, input/output inout declarations 266, node declarations 267, D/T flip flops 268, logic equation assignments 269, and tristate/open drain outputs 270.

[0042] Steps similar to that illustrated in Fig. 7 may be used to generate ABEL code for a corresponding set of boolean equations. Fig. 8D shows the synthesizable and simulatable ABEL code corresponding to the 4-bit counter of Fig. 8A.

[0043] Exemplary listings of programs (written in PERL) that can generate VHDL, Verilog and ABEL codes of the present invention are provided herein as appendix. The listings are recorded on a compact disc as a single text file entitled retarget.txt (created April 22, 2001 and is 84 KB in size). The content of this file is incorporated herein by reference.

[0044] While the invention has been particularly shown and described with reference to a preferred embodiment and several alternate embodiments, it will be understood by persons skilled in the relevant art that various changes in form and detail can be made therein without departing from the spirit and scope of the invention.